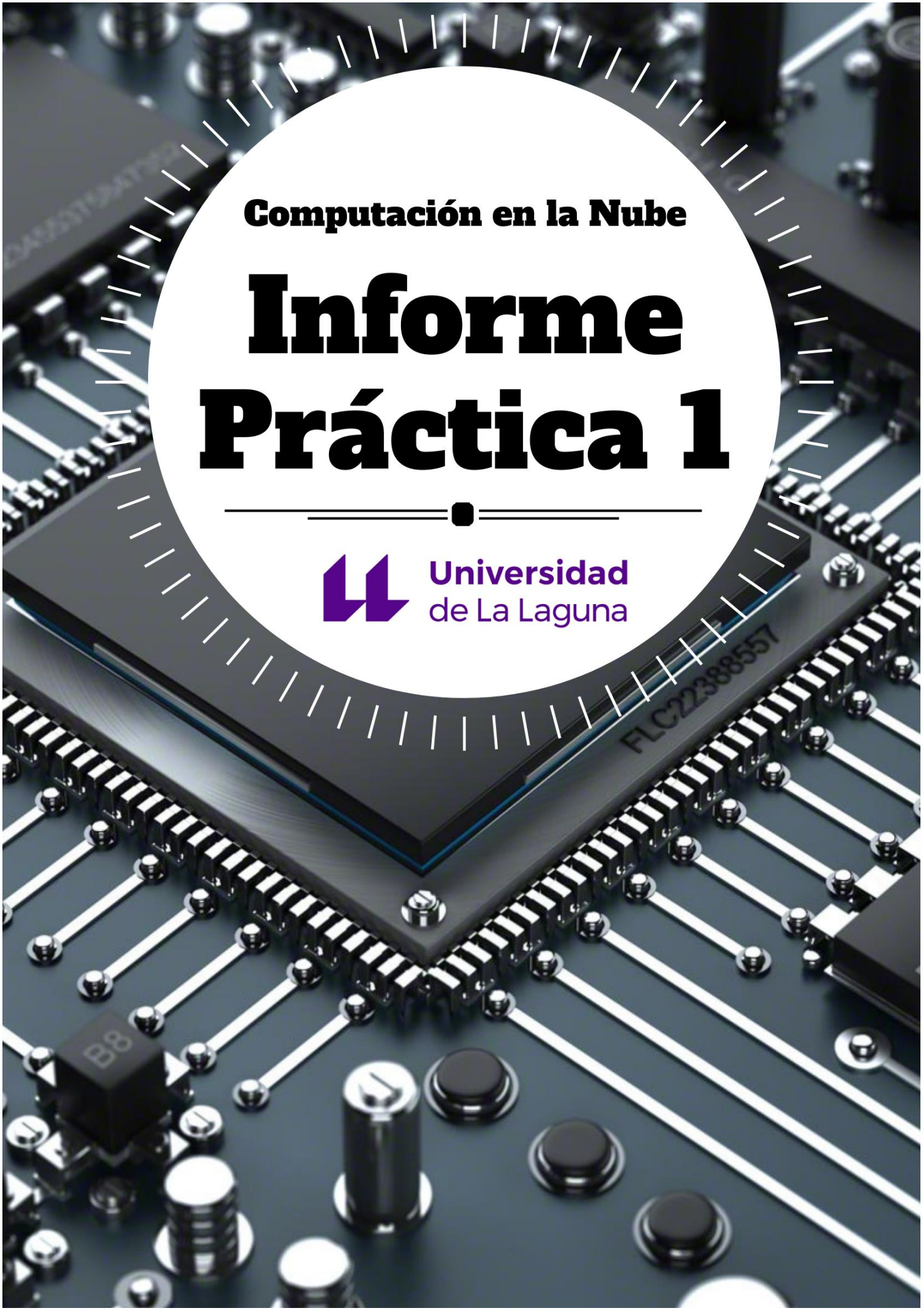

Computación en la Nube
Informe
Práctica 1
Universidad
de La Laguna

Práctica 1

1.- Analiza el programa hello.c y realiza las siguientes ejecuciones comprobando en cada caso el resultado obtenido.

En primera instancia, se ha intentado entender el pequeño fragmento, el cual queda explicado en la siguiente imagen:

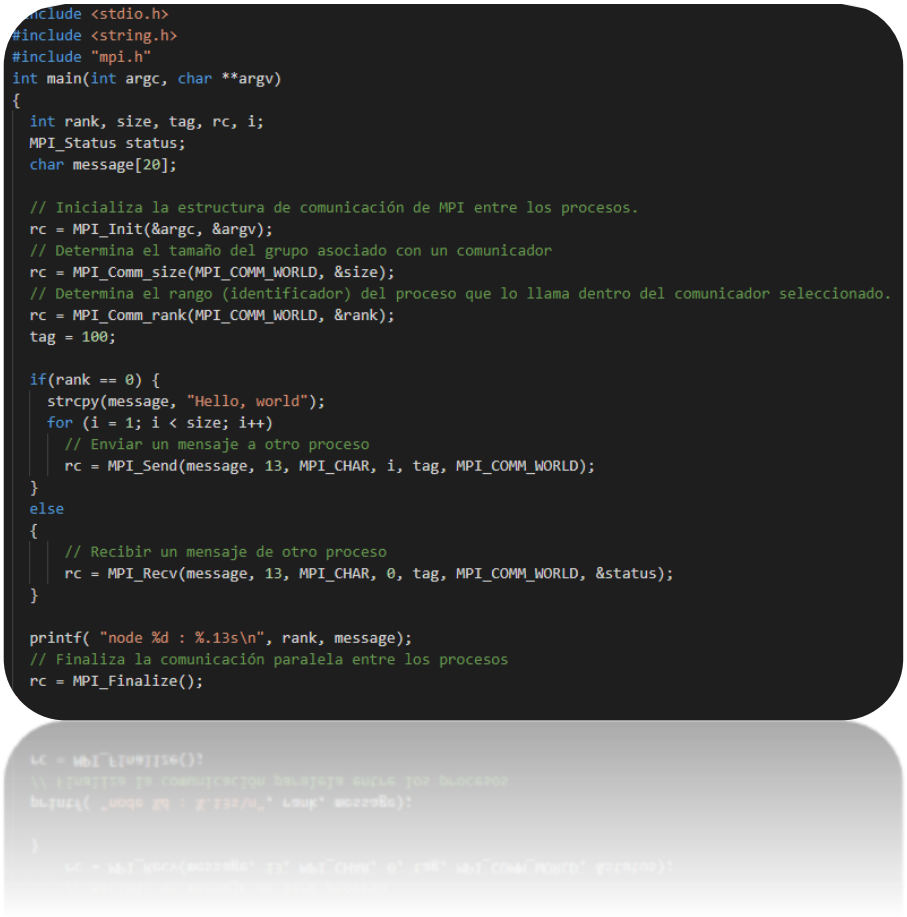
```c
#include <stdio.h>
#include "mpi.h"

int main(int  argc, char **argv)
{

    int rank, size;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    // Inicializa la estructura de comunicación de MPI entre los procesos.
    MPI_Init( &argc, &argv );
    // Determina el tamaño del grupo asociado con un comunicador
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    // Determina el rango (identificador) del proceso que lo llama dentro del comunicador seleccionado.
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    // Obtiene el nombre del proceso
    MPI_Get_processor_name(processor_name, &namelen);

    // Imprime el mensaje del procesos en cuestión
    printf( "Hello world from process %d of %d in %s\n", rank, size, processor_name );

    // Finaliza la comunicación paralela entre los procesos
    MPI_Finalize();
}
```

Como vemos en la imagen anterior este código nos permitirá poder ejecutarlo diversos procesos (tantos como se le pasen por parámetro). Posteriormente, se ha probado el programa con diversos tamaños obtenido la siguiente salida:

```
usuario@lubuntu-1604:~/Descargas$  $mpicc hello.c
usuario@lubuntu-1604:~/Descargas$  $./a.out
Hello world from process 0 of 1 in lubuntu-1604
usuario@lubuntu-1604:~/Descargas$  $mpirun -np 4 ./a.out
Hello world from process 0 of 4 in lubuntu-1604
Hello world from process 1 of 4 in lubuntu-1604
Hello world from process 2 of 4 in lubuntu-1604
Hello world from process 3 of 4 in lubuntu-1604
usuario@lubuntu-1604:~/Descargas$  $mpirun -np 6 ./a.out
Hello world from process 0 of 6 in lubuntu-1604
Hello world from process 1 of 6 in lubuntu-1604
Hello world from process 2 of 6 in lubuntu-1604
Hello world from process 4 of 6 in lubuntu-1604
Hello world from process 3 of 6 in lubuntu-1604
Hello world from process 5 of 6 in lubuntu-1604
usuario@lubuntu-1604:~/Descargas$  $mpirun -np 2 ./a.out
Hello world from process 0 of 2 in lubuntu-1604
Hello world from process 1 of 2 in lubuntu-1604
usuario@lubuntu-1604:~/Descargas$  $
```

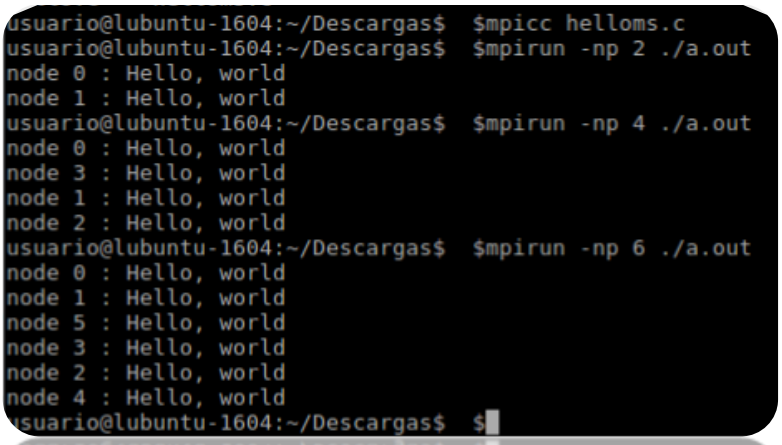
2.- Analiza y compila el programa helloms.

En primera instancia, se ha intentado entender el pequeño fragmento, el cual queda explicado en la siguiente imagen:



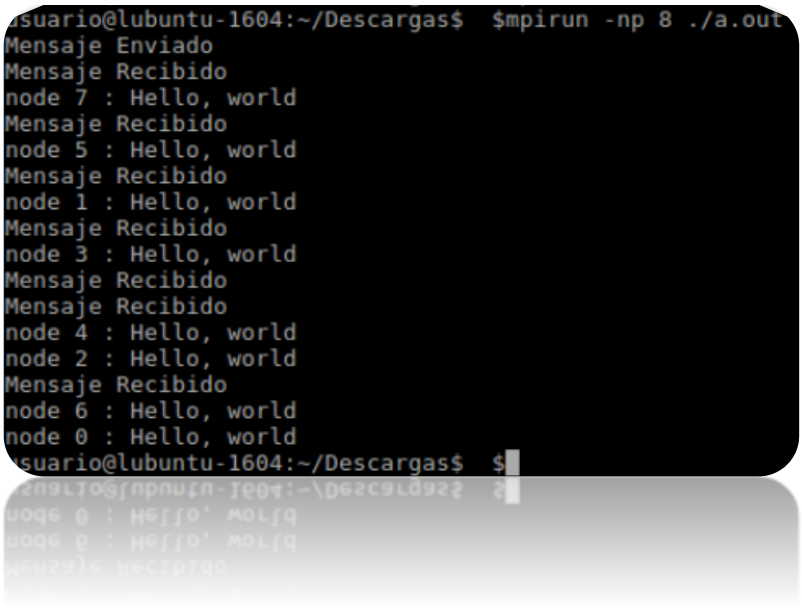
```c
#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main(int argc, char **argv)
{
  int rank, size, tag, rc, i;
  MPI_Status status;
  char message[20];

  // Inicializa la estructura de comunicación de MPI entre los procesos.
  rc = MPI_Init(&argc, &argv);
  // Determina el tamaño del grupo asociado con un comunicador
  rc = MPI_Comm_size(MPI_COMM_WORLD, &size);
  // Determina el rango (identificador) del proceso que lo llama dentro del comunicador seleccionado.
  rc = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  tag = 100;

  if(rank == 0) {
    strcpy(message, "Hello, world");
    for (i = 1; i < size; i++)
      // Enviar un mensaje a otro proceso
      rc = MPI_Send(message, 13, MPI_CHAR, i, tag, MPI_COMM_WORLD);
  }
  else
  {
      // Recibir un mensaje de otro proceso
      rc = MPI_Recv(message, 13, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
  }

  printf( "node %d : %.13s\n", rank, message);
  // Finaliza la comunicación paralela entre los procesos
  rc = MPI_Finalize();
```

Este nos permitirá poder enviar desde un proceso principal un mensaje a todos los restantes (pasados por parámetro). Posteriormente, se ha probado el programa con diversos tamaños obtenido la siguiente salida:



```
usuario@lubuntu-1604:~/Descargas$  $mpicc helloms.c
usuario@lubuntu-1604:~/Descargas$  $mpirun -np 2 ./a.out
node 0 : Hello, world
node 1 : Hello, world
usuario@lubuntu-1604:~/Descargas$  $mpirun -np 4 ./a.out
node 0 : Hello, world
node 3 : Hello, world
node 1 : Hello, world
node 2 : Hello, world
usuario@lubuntu-1604:~/Descargas$  $mpirun -np 6 ./a.out
node 0 : Hello, world
node 1 : Hello, world
node 5 : Hello, world
node 3 : Hello, world
node 2 : Hello, world
node 4 : Hello, world
usuario@lubuntu-1604:~/Descargas$  $
```

A su vez, se ha implementado un mensaje para corroborar lo entendido con anterioridad.

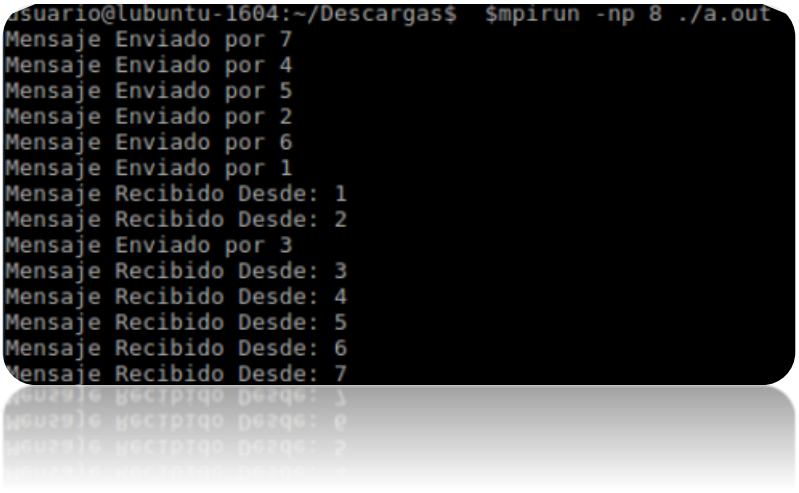


3.- Escribe un nuevo programa en el que los esclavos envían al maestro el mensaje y es el maestro el que muestra la salida.

Para este ejercicio se ha cogido el código proporcionado en el ejercicio 2 añadiendo los siguientes cambios:

```c
#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main(int argc, char **argv)
{
  int rank, size, tag, rc, i;
  MPI_Status status;
  char message[20];

  // Inicializa la estructura de comunicación de MPI entre los procesos.
  rc = MPI_Init(&argc, &argv);
  // Determina el tamaño del grupo asociado con un comunicador
  rc = MPI_Comm_size(MPI_COMM_WORLD, &size);
  // Determina el rango (identificador) del proceso que lo llama dentro del comunicador seleccionado.
  rc = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  tag = 100;

  if(rank == 0) {
    for(i = 1; i < size; i++)
    {
      //El maestro espera por el mensaje de los nodos hijos
      rc = MPI_Recv(message, 13, MPI_CHAR, i, tag, MPI_COMM_WORLD, &status);
      printf( "Mensaje Recibido Desde: %d\n", i);
    }
  }
  else {
    strcpy(message, "Hello, world");
    // Cada nodo hijo envía al maestro
    rc = MPI_Send(message, 13, MPI_CHAR, 0, tag, MPI_COMM_WORLD);
    printf( "Mensaje Enviado por %d\n", rank);
  }

  // Finaliza la comunicación paralela entre los procesos
  rc = MPI_Finalize();
```

Con este fragmento de código se ha obtenido la siguiente salida:

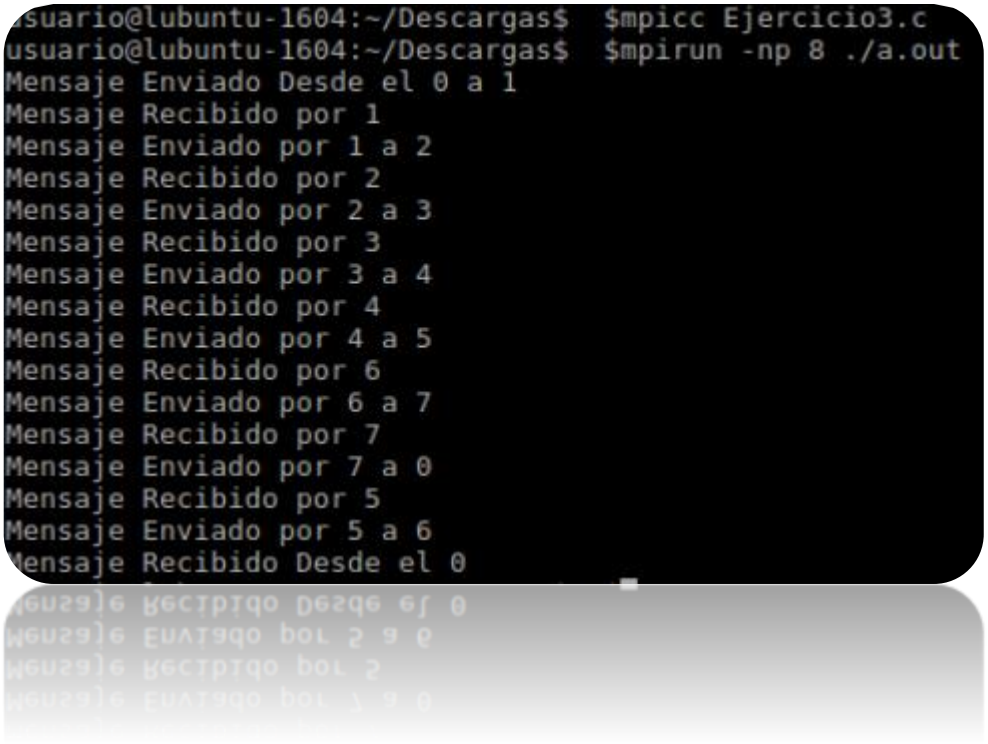


4.- Escribe un programa que haga circular un token en un anillo

Para este programa (al igual que el anterior apartado) se ha escogido el ejemplo propuesto por el profesor y, se ha modificado hasta obtener lo siguiente:

```c
#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main(int argc, char **argv)
{
  int rank, size, tag, rc, i, anterior, posterior;
  MPI_Status status;
  char message[20];

  // Inicializa la estructura de comunicación de MPI entre los procesos.
  rc = MPI_Init(&argc, &argv);
  // Determina el tamaño del grupo asociado con un comunicador
  rc = MPI_Comm_size(MPI_COMM_WORLD, &size);
  // Determina el rango (identificador) del proceso que lo llama dentro del comunicador seleccionado.
  rc = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  tag = 100;

  if(rank == 0) {
     printf( "Mensaje Enviado Desde el 0 a 1\n");
     //Envía el mensaje al siguiente proceso (1 por defecto)
     rc = MPI_Send(message, 13, MPI_CHAR, 1, tag, MPI_COMM_WORLD);

     // Recibe el mensaje del proceso "Size - 1"
     rc = MPI_Recv(message, 13, MPI_CHAR, size - 1, tag, MPI_COMM_WORLD, &status);
     printf( "Mensaje Recibido Desde el 0\n");
  }
  else {
    strcpy(message, "Hello, world");
    anterior = rank - 1;
    // Recibe el mensaje del proceso anterior
    rc = MPI_Recv(message, 13, MPI_CHAR, anterior, tag, MPI_COMM_WORLD, &status);
    printf( "Mensaje Recibido por %d\n", rank);

    if(rank + 1 == size)
    {
      posterior = 0;
    }
    else
    {
      posterior = rank + 1;
    }

    // Envía el mensaje al proceso siguiente
    rc = MPI_Send(message, 13, MPI_CHAR, posterior, tag, MPI_COMM_WORLD);
    printf( "Mensaje Enviado por %d a %d\n", rank, posterior);
  }
  rc = MPI_Finalize();
```

Posteriormente, se ha probado obteniendo la siguiente salida:



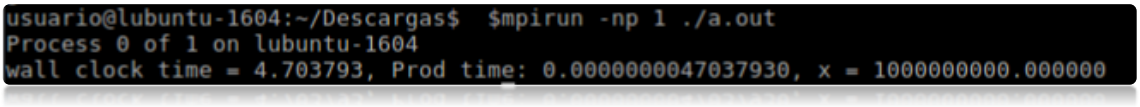
5.- El objetivo de este ejercicio es comprobar experimentalmente el costo de las comunicaciones entre pares de procesadores mediante ping-pong. Se trata además de comparar el coste de las comunicaciones con el coste de hacer una operación de tipo aritmético.

a) Analiza cuál debería ser la salida de los programas prod.c y ptop.c. Compila bajo MPI los programas prod.c y ptop.c. Debes ejecutar el programa prod.c con un único procesador y el programa ptop.c únicamente con dos procesadores.

b) Representa gráficamente la salida que has obtenido con el programa ptop. Utiliza un paquete estadístico o una hoja de cálculo para realizar la regresión lineal de los datos obtenidos con el programa ptop. Representa gráficamente el ajuste y los datos obtenidos experimentalmente.

a) Analiza cuál debería ser la salida de los programas prod.c y ptop.c. Compila bajo MPI los programas prod.c y ptop.c. Debes ejecutar el programa prod.c con un único procesador y el programa ptop.c únicamente con dos procesadores.

El fichero "prod.c" nos proporciona el tiempo que ha tardado por realizar cada operación. Su salida es la siguiente:



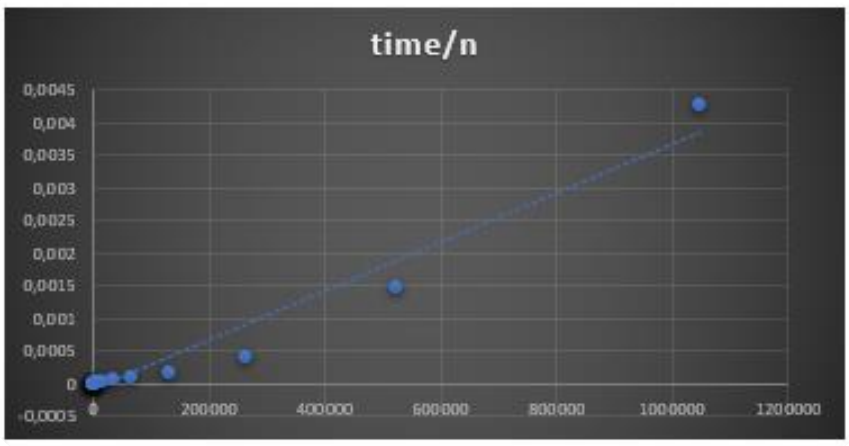
El fichero “ptop.c” nos proporciona el tiempo que ha tardado en realizar una comunicación con otro proceso. Su salida es la siguiente:

```
uario@lubuntu-1604:~/Descargas$  $mpirun -np 2 ./a.ou

Procesador: lubuntu-1604
Procesador: lubuntu-1604
Kind            n       time (sec)      MB / sec
Send/Recv       1       0.000000        25.980977
Send/Recv       2       0.000000        52.143639
Send/Recv       4       0.000000        104.530941
Send/Recv       8       0.000000        174.308738
Send/Recv       16      0.000000        352.232768
Send/Recv       32      0.000000        610.752230
Send/Recv       64      0.000000        1039.104991
Send/Recv       128     0.000001        1582.356372
Send/Recv       256     0.000001        2061.584302
Send/Recv       512     0.000002        2021.161080
Send/Recv       1024    0.000003        3272.356035
Send/Recv       2048    0.000004        4042.322161
Send/Recv       4096    0.000007        4739.274258
Send/Recv       8192    0.000011        5726.623061
Send/Recv       16384   0.000021        6108.397932
Send/Recv       32768   0.000044        6024.721248
Send/Recv       65536   0.000078        6724.841760
Send/Recv       131072  0.000164        6411.146518
Send/Recv       262144  0.000402        5217.137024
Send/Recv       524288  0.001459        2875.714923
nd/Recv         1048576 0.004257        1970.340600
```

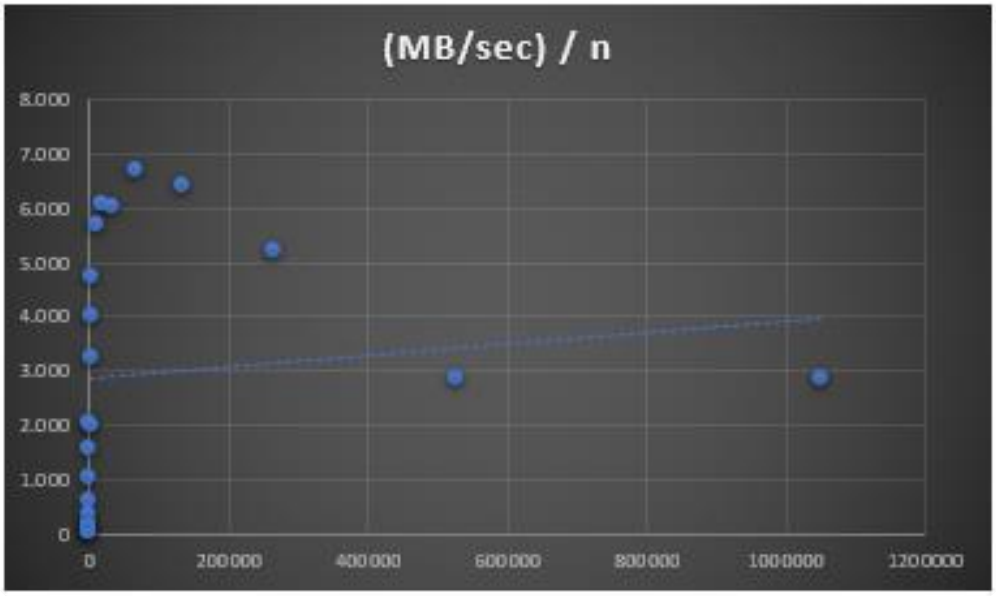
b) Representa gráficamente la salida que has obtenido con el programa ptop. Utiliza un paquete estadístico o una hoja de cálculo para realizar la regresión lineal de los datos obtenidos con el programa ptop. Representa gráficamente el ajuste y los datos obtenidos experimentalmente.

Las gráficas de regresión lineal generadas mediante Excel han sido las siguientes:

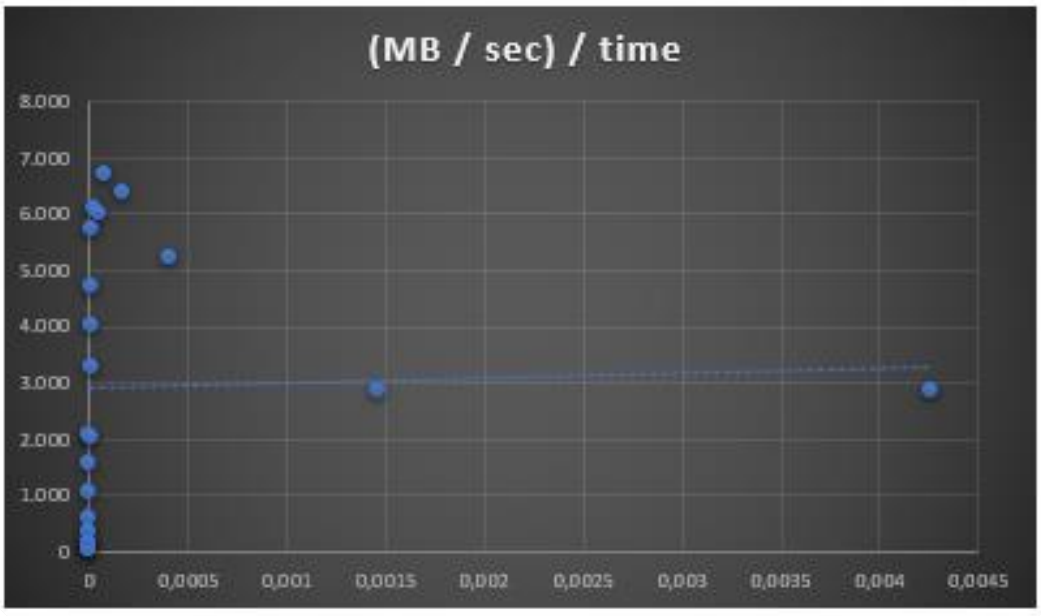
Tiempo/ Número de comunicaciones



(MB/sec)/ Número de comunicaciones



(MB/sec)/ tiempo de comunicación



La ecuación de la recta de la gráfica (time/n) anterior viene definida por la siguiente ecuación:

$$y=0.000000003754515285 96x+-0.000067721468147627 36$$

Como hemos visto en la práctica, tenemos que tener en cuenta cuando es rentable realizar una comunicación (teniendo en cuenta el coste para el tamaño del dato) y cuando es más rentable no realizar la comunicación (hacerlo con el mismo proceso). Esto nos ayuda a cambiar la visión de que cuantos más procesadores mejor, es decir, no siempre nos saldrá mas rentable tener más procesadores y sobrecargar la comunicación. De igual forma, tampoco es lo más rentable darle toda la carga a un solo procesador. En mi opinión, hay que buscar un equilibrio entre ambas casuísticas, sin olvidar el apartado económico.